Table of Contents

The HttpFoundation Component

Before diving into the framework creation process, let's first step back and let's take a look at why you would like to use a framework instead of keeping your plain-old PHP applications as is. Why using a framework is actually a good idea, even for the simplest snippet of code and why creating your framework on top of the Symfony components is better than creating a framework from scratch.

> We won't talk about the obvious and traditional benefits of using a framework when working on big applications with more than a few developers; the Internet has already plenty of good resources on that topic.

Even if the "application" we wrote in the previous chapter was simple enough, it suffers from a few problems:

```
// framework/index.php
$name = $_GET['name'];

printf('Hello %s', $name);
```

First, if the `name` query parameter is not defined in the URL query string, you will get a PHP warning; so let's fix it:

```
// framework/index.php
$name = isset($_GET['name']) ? $_GET['name'] : 'World';

printf('Hello %s', $name);
```

Then, this *application is not secure*. Can you believe it? Even this simple snippet of PHP code is vulnerable to one of the most widespread Internet security issue, XSS (Cross-Site Scripting). Here is a more secure version:

```
$name = isset($_GET['name']) ? $_GET['name'] : 'World';

header('Content-Type: text/html; charset=utf-8');

printf('Hello %s', htmlspecialchars($name, ENT_QUOTES, 'UTF-8'));
```

As you might have noticed, securing your code with `htmlspecialchars` is tedious and error prone. That's one of the reasons why using a template engine like Twig (https://twig.symfony.com/), where auto-escaping is enabled by default, might be a good idea (and explicit escaping is also less painful with the usage of a simple `e` filter).

As you can see for yourself, the simple code we had written first is not that simple anymore if we want to avoid PHP warnings/notices and make the code more secure.

Beyond security, this code is not even easily testable. Even if there is not much to test, it strikes me that writing unit tests for the simplest possible snippet of PHP code is not natural and feels ugly. Here is a tentative PHPUnit unit test for the above code:

```
// framework/test.php
use PHPUnit\Framework\TestCase;

class IndexTest extends TestCase
{
    public function testHello()
    {
        $_GET['name'] = 'Fabien';

        ob_start();
        include 'index.php';
        $content = ob_get_clean();

        $this->assertEquals('Hello Fabien', $content);
    }
}
```

If our application were just slightly bigger, we would have been able to find even more problems. If you are curious about them, read the *Symfony versus Flat PHP* (../introduction/from_flat_php_to_symfony2.html) chapter of the book.

At this point, if you are not convinced that security and testing are indeed two very good reasons to stop writing code the old way and adopt a framework instead (whatever adopting a framework means in this context), you can stop reading this book now and go back to whatever code you were working on before.

Of course, using a framework should give you more than just security and testability, but the more important thing to keep in mind is that the framework you choose must allow you to write better code faster.

Going OOP with the HttpFoundation Component ¶

Writing web code is about interacting with HTTP. So, the fundamental principles of our framework should be around the HTTP specification (https://tools.ietf.org/wg/httpbis/).

The HTTP specification describes how a client (a browser for instance) interacts with a server (our application via a web server). The dialog between the client and the server is specified by well-defined *messages*, requests and responses: *the client sends a request to the server and based on this request, the server returns a response*.

In PHP, the request is represented by global variables (`$_GET`, `$_POST`, `$_FILE`, `$_COOKIE`, `$_SESSION`...) and the response is generated by functions (`echo`, `header`, `setcookie`, ...).

The first step towards better code is probably to use an Object-Oriented approach; that's the main goal of the Symfony HttpFoundation component: replacing the default PHP global variables and functions by an Object-Oriented layer.

To use this component, add it as a dependency of the project:

```
$ composer require symfony/http-foundation
```

Running this command will also automatically download the Symfony HttpFoundation component and install it under the `vendor/` directory. A `composer.json` and a `composer.lock` file will be generated as well, containing the new requirement.

Class Autoloading

When installing a new dependency, Composer also generates a `vendor/autoload.php` file that allows any class to be easily autoloaded (https://php.net/autoload). Without autoloading, you would need to require the file where a class is defined before being able to use it. But thanks to PSR-4 (https://www.php-fig.org/psr/psr-4/), we can just let Composer and PHP do the hard work for us.

Now, let's rewrite our application by using the `Request` and the `Response` classes:

```
// framework/index.php
require_once __DIR__.'/vendor/autoload.php';

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

$request = Request::createFromGlobals();

$name = $request->get('name', 'World');

$response = new Response(sprintf('Hello %s', htmlspecialchars($name, ENT_QUOTES, 'UTF-8')))

$response->send();
```

The `createFromGlobals()` method creates a `Request` object based on the current PHP global variables.

The `send()` method sends the `Response` object back to the client (it first outputs the HTTP headers followed by the content).

Before the `send()` call, we should have added a call to the `prepare()` method (`$response->prepare($request);`) to ensure that our Response were compliant with the HTTP specification. For instance, if we were to call the page with the `HEAD` method, it would remove the content of the Response.

The main difference with the previous code is that you have total control of the HTTP messages. You can create whatever request you want and you are in charge of sending the response whenever you see fit.

We haven't explicitly set the `Content-Type` header in the rewritten code as the charset of the Response object defaults to `UTF-8`.

With the `Request` class, you have all the request information at your fingertips thanks to a nice and simple API:

You can also simulate a request:

```
$request = Request::create('/index.php?name=Fabien');
```

With the `Response` class, you can easily tweak the response:

To debug a response, cast it to a string; it will return the HTTP representation of the response (headers and content).

Last but not least, these classes, like every other class in the Symfony code, have been audited (https://symfony.com/blog/symfony2-security-audit) for security issues by an independent company. And being an Open-Source project also means that many other developers around the world have read the code and have already fixed potential security problems. When was the last time you ordered a professional security audit for your home-made framework?

Even something as simple as getting the client IP address can be insecure:

```
if ($myIp === $_SERVER['REMOTE_ADDR']) {
    // the client is a known one, so give it some more privilege
}
```

It works perfectly fine until you add a reverse proxy in front of the production servers; at this point, you will have to change your code to make it work on both your development machine (where you don't have a proxy) and your servers:

```
if ($myIp === $_SERVER['HTTP_X_FORWARDED_FOR'] || $myIp === $_SERVER['REMOTE_ADDR']) {
    // the client is a known one, so give it some more privilege
}
```

Using the `Request::getClientIp()` method would have given you the right behavior from day one (and it would have covered the case where you have chained proxies):

```
$request = Request::createFromGlobals();

if ($myIp === $request->getClientIp()) {
    // the client is a known one, so give it some more privilege
}
```

And there is an added benefit: it is *secure* by default. What does it mean? The `$_SERVER['HTTP_X_FORWARDED_FOR']` value cannot be trusted as it can be manipulated by the end user when there is no proxy. So, if you are using this code in production without a proxy, it becomes trivially easy to abuse your system. That's not the case with the `getClientIp()` method as you must explicitly trust your reverse proxies by calling `setTrustedProxies()`:

```
Request::setTrustedProxies(array('10.0.0.1'));

if ($myIp === $request->getClientIp()) {
    // the client is a known one, so give it some more privilege
}
```

So, the `getClientIp()` method works securely in all circumstances. You can use it in all your projects, whatever the configuration is, it will behave correctly and safely. That's one of the goal of using a framework. If you were to write a framework from scratch, you would have to think about all these cases by yourself. Why not using a technology that already works?

If you want to learn more about the HttpFoundation component, you can have a look at the `HttpFoundation (https://api.symfony.com/4.1/Symfony/Component/HttpFoundation.html)` API or read its dedicated *documentation* (../components/http_foundation.html).

Believe or not but we have our first framework. You can stop now if you want. Using just the Symfony HttpFoundation component already allows you to write better and more testable code. It also allows you to write code faster as many day-to-day problems have already been solved for you.

As a matter of fact, projects like Drupal have adopted the HttpFoundation component; if it works for them, it will probably work for you. Don't reinvent the wheel.

I've almost forgot to talk about one added benefit: using the HttpFoundation component is the start of better interoperability between all frameworks and applications using it (like Symfony (https://symfony.com/), Drupal 8 (https://drupal.org/), phpBB 3 (https://www.phpbb.com/), ezPublish 5 (https://ez.no/), Laravel (https://laravel.com/), Silex (https://silex.sensiolabs.org/) and more (https://symfony.com/components/HttpFoundation)).

Latest from the Symfony Blog

SymfonyLive Berlin 2018 Call for Papers (/blog/symfonylive-berlin-2018-call-for-papers)
July 18, 2018

They Help Us Make Symfony

Upcoming official conferences

SymfonyLive London 2018 (https://london2018.live.symfony.com/)
London (United Kingdom) - September 27-28, 2018

SymfonyLive USA 2018 (https://usa2018.live.symfony.com/)
San Francisco, CA (United States) - October

Upcoming training sessions

Getting Started with Symfony 3 on Blended Learning (https://training.sensiolabs.com/en/courses?q=SF3C1&from=07/23/2018&to=07/24/2018)
Paris - July 23, 2018

Web Development with Symfony 3 (https://training.sensiolabs.com

SensioLabsWorld

Security Monitoring: PHP security vulnerabilities monitoring (https://security.symfony.com/)

Connect (/session/new?target=https://symfony.com/doc/current/create_framework/http_foundation.html)

Thanks Michael Sivolobov (https://connect.sensiolabs.com/profile/astronomer) for being a Symfony contributor (/contributors).

2 commits · 4 lines

11–12, 2018

SymfonyLive Berlin 2018 (http://berlin2018.live.symfony.com)

Berlin (Germany) - October 24–26, 2018

SymfonyCon Lisbon 2018 (https://lisbon2018.symfony.com/)

Lisbon (Portugal) - December 6–8, 2018

/en/courses?q=SF3C4&from=07/23/2018&to=07/26/2018)

Paris - July 23, 2018

Mastering Symfony 3 on Blended Learning (https://training.sensiolabs.com/en/courses?q=SF3C2&from=07/25/2018&to=07/26/2018)

Paris - July 25, 2018

View all sessions →

SensioLabsWorld Security Monitoring: PHP security vulnerabilities monitoring (https://security.symfony.com/)

Connect (/session/new?target=https://symfony.com/doc/current/create_framework/http_foundation.html)

What is Symfony? (/what-is-symfony)
Symfony at a Glance (/at-a-glance)
Symfony Components (/components)
Case Studies (/blog/category/case-studies)
Symfony Roadmap (/roadmap)
Security Policy (/doc/current/contributing/code/security.html)
Logo & Screenshots (/logo)
Trademark & Licenses (/license)
symfony1 Legacy (/legacy)

Learn Symfony (/doc/current/index.html)
Getting Started (/doc/current/setup.html)
Components (/doc/current/components/index.html)
Best Practices (/doc/current/best_practices/index.html)
Bundles (/doc/bundles/)
Reference (/doc/current/reference/index.html)
Training (https://training.sensiolabs.com/en/courses?q=symfony)
Certification (https://certification.symfony.com/)

Community (/community)
SensioLabs Connect (https://connect.sensiolabs.com/)
Support (/support)
How to be Involved (/doc/current/contributing/index.html)
Events & Meetups (/events/)
Projects using Symfony (/projects)
Code Stats (/stats/code)
Downloads Stats (/stats/downloads)
Contributors (/contributors)

Blog (/blog/)
A week of symfony (/blog/category/a-week-of-symfony)
Case studies (/blog/category/case-studies)
Community (/blog/category/community)
Conferences (/blog/category/conferences)
Diversity (/blog/category/diversity)
Documentation (/blog/category/documentation)
Living on the edge (/blog/category/living-on-the-edge)
Releases (/blog/category/releases)
Security Advisories (/blog/category/security-advisories)

Services (https://sensiolabs.com)
Our services (https://sensiolabs.com)
Train developers (https://training.sensiolabs.com/en)
Manage your project quality (https://insight.sensiolabs.com/)
Improve your project performance (https://blackfire.io/)

About (/about)
SensioLabs (https://sensiolabs.com/en/join_us/join_us.html)
Contributors (/contributors)
Careers (https://sensiolabs.com/en/join_us/join_us.html)
Support (/support)

Follow Symfony

(https://github.com/symfony) (https://stackoverflow.com/questions/tagged/symfony) (/slack-invite) (https://twitter.com/symfony) (https://www.facebook.com/SymfonyFramework) (https://www.youtube.com/user/SensioLabs) (https://medium.com/@symfony)